RISC-V Implementation

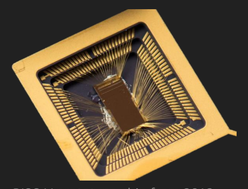
Author: Jack Parkinson **Student#:** U1552044

Supervisor: Dr Violeta Holmes

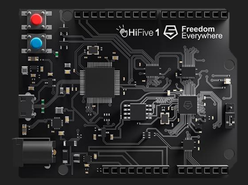
Course: Electronic Engineering and Computer Systems BEng(Hons)

Background

- Traditionally microprocessors fall into two categories, **CISC** (Complex Instruction Set Computer) or **RISC** (Reduced Instruction Set Computer)
- Most CISC designs now utilise microoperations meaning they're essentially RISC designs at their lowest level
- More application specific microprocessor designs are needed to meet the needs of emerging technologies. However, economies of scale do not work in favour of these designs
- In an attempt to alleviate economies of scale RISC-V aims to replicate the Open Source model at the hardware level by providing a free and open **ISA** (Instruction Set Architecture)
- A proven method for gaining loyal community members is to use an open source project as an educational tool



RISC-V prototype chip from 2013



RISC-V development board 2018

Aim

The aim of this project is to design and implement a soft RISC-V core that could be used as an educational tool in computer architecture and organisation

Objectives

- ✓ Design a soft processor core capable of executing Integer Computation, Load/Store and Control Transfer instructions
- ✓ Implement the soft processor core using VHDL for use in FPGAs
- ✓ Test the implementation in simulation
- ✖ Test the implementation on an FPGA development board

Poster References

- RISC-V Foundation. (2017). Specifications. Retrieved from https://riscv.org/specifications/.
- Tilley, A.T. (2016,). This New Chip Startup Wants To Bring Open Source To A Stagnant Industry. Retrieved from https://www.forbes.com/sites/aarontilley/2016/07/11/risc-v-sifive-chips/#62eee3c729ec.
- SiFive. (N.d). Boards & Software. Retrieved from https://www.sifive.com/boards.

Instruction Set Architecture

The design adheres to a subset of the RV32I v2.0 base integer instruction set defined in the RISC-V user level specification. The main characteristics of this instruction set are:

- Fixed-length 32-bit instructions, aligned on a four-byte boundary in memory
- Has 31 general purpose registers x1-x31 with one register x0 hardwired to the constant 0
- Uniform decoding, split into four core formats (R/I/S/U) and two sub formats (B/J) which are variants of (S/U) respectively

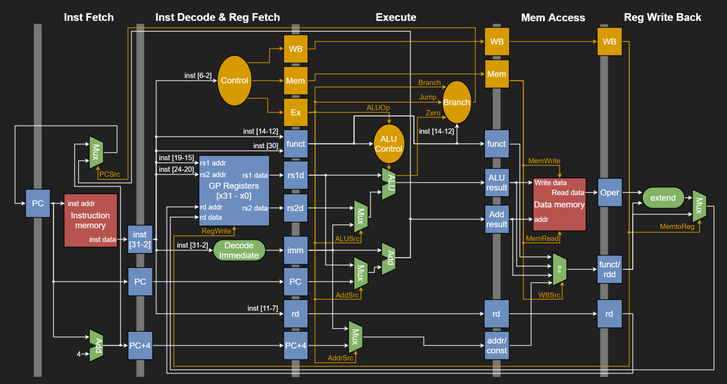
31 30 25	24 21	20 19 15	14 12	11 8	7 6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12] imm[10:5]	rs2	rs1	funct3	imm[4:1] imm[11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20] imm[10:1]		imm[11]	imm[19:12]	rd	opcode	J-type

- Little-Endian with the sign bit for all immediates held in bit 31 of the instruction, allowing sign-extension to proceed in parallel with instruction decoding

31 30 20	19 12	11 10 5	4 1	0	
- inst[31]-		inst[30:25]	inst[24:21]	inst[20]	I-imm
- inst[31]-		inst[30:25]	inst[11:8]	inst[7]	S-imm
- inst[31]-	inst[7]	inst[30:25]	inst[11:8]	0	B-imm
inst[31] inst[30:20]	inst[19:12]	-0-			U-imm
-inst[31]-	inst[19:12]	inst[20]	inst[30:25]	inst[24:21]	0 J-imm

Core

- Critical path design that prioritised performance over size
- Classic 5-Stage RISC Pipeline
- Byte-aligned memory that supports misaligned access

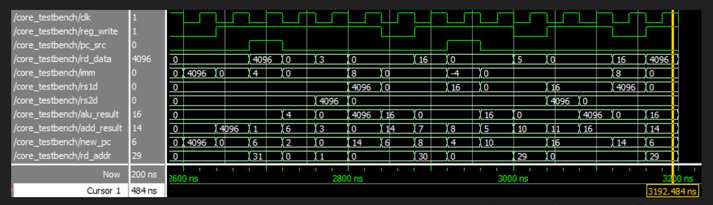


Opcode	Control								
	RegWrite	MemtoReg & MemRead	MemWrite	WBSrc[1]	WBSrc[0] & ALUOp[1]	AddrSrc & ALUSrc	AddSrc & ALUOp[0]	Branch	Jump
LUI	1	0	0	0	0	1	X	0	0
AUIPC	1	0	0	0	1	X	0	0	0
OP-IMM	1	0	0	1	0	0	0	0	0
OP	1	0	0	1	0	1	1	0	0
JAL	1	0	0	0	0	0	0	0	1
JALR	1	0	0	0	0	0	1	0	1
BRANCH	0	0	0	X	1	1	0	1	0
LOAD	1	1	0	1	1	X	1	0	0
STORE	0	0	1	X	1	1	1	0	0

WB Mem Ex

Compiler & Results

Each VHDL entity was tested in RTL level simulation using custom testbenches also written in VHDL. To test the core as a whole, a testbench was created which acted as an assembly compiler and uploaded the compiled binary into the core's instruction memory




```
-- 0
lui(31, 1, signals); -- x31 = 4096
-- 1
nop(signals);
-- 2
jal(1, 2, signals); -- PC = 6
-- 3
nop(signals);
-- 4
sw(30, 31, 0, signals); -- data_addr[16 + 0] = 4096 (data_addr[rs1 + imm] = rs2)
-- 5
lw(29, 30, 0, signals); -- x29 = data_addr[16 + 0] (data_addr[rs1 + imm] = rs2)
-- 6
srli(30, 31, 8, signals); -- x30 = 4096 >> 8 (x31 >> constant) = 16
-- 7
nop(signals);
-- 8
nop(signals);
-- 9
bne(30, 31, -2, signals); -- if x30 != x31, PC = 4
```

A variety of assembly programs were run to test every instruction. This included one program which aimed to test every type of instruction acting on a single operand so any errors would affect the final result

Conclusion

The soft RISC-V core design manages to demonstrate many implementation techniques while still being relatively simple in its overall design. The VHDL implementation correctly executes all of the RV32I instructions it was designed to. The combination of the design, implementation and technical report could be used as an educational tool meaning this project can be deemed a success. However, given the additional time and resources it could greatly benefit from:

- Ensuring that the VHDL implementation can be used on FPGA by:
  1. Completing timing simulation (50MHz target on Cyclone IV)
  2. Testing on an FPGA development board
- Making the design and VHDL implementation compatible with existing open source compilers by:
  1. Adding stall/flush logic & forwarding unit to the design to prevent data hazards
  2. Adding logic to handle RISC-V hardware threads (harts) to the design, allowing it to execute the full RV32I instruction set
- Increasing the design's effectiveness as an education tool by breaking it up into smaller more easily digestible designs